Washcloth Simulation of Three-Dimensional Weather Forcasting Codes

by

David Korn
Norman Rushfield

Ultracomputer Note #55
May, 1983

Ultracomputer Research Laboratory

New York University
Courant Institute of Mathematical Sciences
Division of Computer Science
251 Mercer Street, New York, NY 10012

Washcloth Simulation of Three-Dimensional Weather Forcasting Codes

by

David Korn
Norman Rushfield

Ultracomputer Note #55
May, 1983

This work was supported in part by NASA under grant No. NSG-5034, by the NSF under grant No. NSF-MCS79-07804, and by the DOE under grant No. AC02-76ER03077.

Ultracomputer Note #55

WASHCLOTH SIMULATION OF THREE-DIMENSIONAL WEATHER FORCASTING CODES

David Korn
Norman Rushfield
May 10, 1983

Chapter I

INTRODUCTION

This note describes the method used, and the results obtained for simulating an existing weather forcasting program of Rivas[6] with the WASHCLOTH[1,2] simulator. The WASHCLOTH simulator is used to investigate how FORTRAN programs would behave on an idealized MIMD machine, the paracomputer[12], which will be approximated by the NYU Ultracomputer[5].

The memo sheds light on the problems encountered in taking existing codes and running them on the Ultracomputer.

This work was supported in part by NASA under grant NSG-5034, by NSF under grant number NSF-MCS79-07804, and by DOE under grant DE-AC02-76ER03077.

CHAPTER 2

PROBLEM DESCRIPTION

An earlier paper[11] reported on the simulation of a two-dimensional atmospheric model code for weather forcasting. The success of that project was the impetus for taking a complete three-dimensional atmospheric model code used for weather forcasting, and simulating it under WASHCLOTH. Some details of this model are reproduced in the Appendix from reference [6].

The code we received from NASA/Goddard, was written in FORTRAN for an AMDAHL computer. The code consisted of more than 7000 lines and used most of the 16M-bytes of memory present in the AMDAHL.

The code generates solutions to a global circulation model of the atmosphere by a finite difference scheme which computes values of several interrelated state variables on a three-dimensional grid encircling the globe. The equations of motion are listed in the Appendix. A region of fixed altitude is called a layer. The model uses a fairly small set of layers which is usually a fixed property of the model. The code we received was a nine-layer model. The grid on each layer is

uniformly distributed in latitude and longitude.

After a given number of iterations, the solution is used to recompute certain physical quantities at each grid point. These computations are algebraic in nature and can clearly be performed at each grid point in parallel. It was decided to remove this part of the code from our study, since we felt that parallelizing it would be easy and would not yield any new information. Furthermore, the omitted code would be very efficient so that the estimates of efficiency we report are conservative.

Given initial data for the state variables, their values at a future time are determined by this program. The difference equations are used to march ahead in time. For stability reasons, the solution method being explicit, the size of a time step is limited by the spatial distance between adjacent points on the coordinate grid system. Thus, to compute the weather at a specific time, more iterations have to be performed when using finer grids and hence, for a fixed number of layers, refining the grid by a factor of two increases the serial computing time by a factor of eight.

Special problems occur near the poles since the grid is evenly spaced in latitude and longitude and hence the spacial distance between adjacent polar grid points is small and instabilities would result if the time step were not correspondingly small. The larger time step actually used, results in amplification of the high frequency harmonics of the discretization error. Smoothing of the solution near the polar

region, which decreases the amplitude of high frequency harmonics, allows for larger time steps without instabilities.

The NASA code offers a choice of two smoothing routines for use in the polar regions. One method uses Fourier filtering to dampen the higher frequency harmonics. The other method uses weighted averages of neighboring points. In both of these methods, the smoothing is only performed on lines of latitude near the poles. The degree of smoothing increases monotonically towards the polar regions.

CHAPTER 3

GETTING THE SERIAL CODE TO RUN

Since WASHCLOTH runs on the CDC-6600, and not on the AMDAHL, it was first necessary to get the serial code running on the CDC-6600. This proved to be the most difficult part of the whole simulation process. The CDC-6600 has a very small memory by present standards, 128K words or about 1M-byte. Obviously, it was necessary to run the program with much smaller grid sizes. Changing grid sizes proved to be non-trivial because many of the arrays were equivalenced to save space. Cutting the array dimensions made some of these equivalences impossible. Also, the initial data had to be generated for points on the reduced grid.

In addition to the severe space problem, there were incompatibilities in the versions of FORTRAN on the two machines. The FTN compiler on the CDC-6600 allows only one, two, or three dimensional arrays. The program we converted used four dimensional arrays. The fourth dimension was used to group four variables rather than having a separate three-dimensional array for each of them.

The initial data had do be interpolated to get values at our grid points. When the program was running on the CDC-6600, there was no way to verify that it had been done correctly. The grid was too crude to generate realistic solutions. Once we were able to verify that all portions of the code were exercised and did not generate obviously erroneous results we certified this code as the CDC-6600 serial code that we were to parallelize and simulate. The results of the parallel code had to agree with this benchmark serial code.

CHAPTER 4

PARALLELIZING THE CODE

Our plan was to use the DOWAIT[9] construct to parallelize the code whenever possible. This construct allows one and two dimensional DO loops for which the calculations for each of the values of the loop indices are independent to be performed simultaneously. Ssynchronization occurs at the end of each loop. The code outside DOWAIT regions is executed by one processing element, PE, whereas inside the DOWAIT, as many processing elements as are available are used as assistants.

The DOWAIT code can be run with or without MOP[3]. When used without MOP, each PE except the one running the serial code, busy waits outside of DOWAIT loops. The DOWAIT code opens a "gate" for these PE's and within a few instructions they perform independent loop iterations.

When using MOP, one PE executes the serial code and the other PE's are available for other jobs rather than busy waiting. Upon entering the DOWAIT, the processor makes an operating system request for helpers. As the loop iterations complete, each of these helpers complete and is released for scheduling by the

operating system. The number of cycles to initiate a helper is substantially higher when using MOP, but no busy waiting is needed while executing serial code or for performing synchronization.

Several changes were required to convert the serial code to a form where it consisted of segments suitable for the DOWAIT construct. The code consistently used index names starting with I for longitude, J for latitude, and L for altitude, and we will use this convention freely as well.

Parts of code required the solution at one value of L to generate the value for L+1, primarily because of quadrature formulas. The values for I and J, at a fixed L, could be determined from values of state variables from the previous iteration, except for the smoothings. If the code were organized as a sequence of loops with the altitude calculations in the inner-most loop, then each of the two outer loops for I and J could be replaced by a sequence of DOWAIT loops ranging over both latitude and longitude. Optimizations performed by serial compilers could still be applied to the inner L loop. Note that this organization, placing serial loops inside parallel loops, is the reverse of that which is suitable for vector processing.

The original code does not have this consistent organization. The organization varied from place to place. This was primarily because certain loop orderings require more temporary space than others. The loops were rearranged as described above whenever possible. The resulting code was

checked against the serial version.

A technique frequently referred to as "banding" was used in the original code to save space for temporary variables. Intermediate results were computed and saved in a band which changes as the iteration proceeds. In this code, banding was done on J so that several temporaries variables were saved for J, J-1, and J-2 only. The solution is computed sequentially from J=1 to J=JMAX and each time J is incremented the J-2 part of the temporary matrices are replaced by J-1, the J-1 elements replaced by J, and the new J elements are computed.

The banding technique saves memory but introduces non-essential serialization which requires that latitudes be done sequentially. In the parallel version of the code, banding was replaced by two DOWAIT loops. The first loop computed and stored all the temporary values at all the grid points. The second performed the updates.

The smoothing portions were also parallelized. In this case, the smoothing for each latitude and each layer were done simultaneously. The amount of computation varies with latitude but this is not a problem since no assumption is made that each sub-task be of the same size. Since the number of layers is small, the degree of parallelism is less here than other parts of the program. For Fourier filtering, several PE's could have been assigned to each of these sub-tasks[10]. This was not done because the filtering represented a relatively small portion of the computing time and limitations of the WASHCLOTH emulator made

nested DOWAIT's rather difficult.

CHAPTER 5

RESULTS

Because of the small memory size of the CDC-6600, it was not possible to simulate realistic grid sizes. The grid size was set to 24x16x3 and simulation runs were made for varying number of processing elements. Table 1 shows the timing results for performing one iteration of the code without MOP. The first entry shows the number of instructions for the serial version of the code.

Each iteration consisted of 55 DOWAIT loops. At the grid that was used, this took 12,208 loop instances. The increase in of about 700,000 instructions for the parallel version is almost exclusively due to the extra instructions involved in selecting a loop iteration. In this case there are about 50 instructions required to initiate a loop iteration. Part of the reason that it took 50 instructions to initiate an iteration is that a general purpose subroutine was called rather than having a compiler that could generate inline code. On the other hand, had we run under MOP, this number might have been substantially higher. Since the time to initiate a DOWAIT/DOALL iteration depends on the implementation, it is helpful to see how the

results depend on the number of instructions needed to initiate an iteration. Table 2 shows how the number of instructions for loop initialization effects efficiency. Since the number of instructions per loop iteration increases as we increase the number of layers, the efficiency would increase with more layers. Since we simulated only a three layer atmospheric model, our results are conservative.

The version of WASHCLOTH used for the simulations allowed only 32 PE's to be simulated. The methodology needed to extrapolate these results to larger grids and more processing elements was presented in reference [8]. Since the computation is dominated by two dimensional DOWAIT loops, doubling the number of latitude and longitude grid points should allow four times as many processing elements to be used with the same efficiencies as in Table 1. Increasing the number of layers would increase the number of instructions in many of the DOWAIT loops and therefore should also improve efficiency substantially. By scaling up to realistic grid sizes, these results can be used to predict the behavior of a system of 4096 processing elements. Runs commonly made today, use grid sizes with 12 times as many points around the globe and three times the number of layers, so that about 500 processing elements could in that case be used higher efficiency than reported here for 32 processing elements. With finer grid sizes all 4096 processing elements could be used to achieve speedups of greater than 3000 over the speed of sequential execution.

THREE-DIMENSIONAL WEATHER CODE

24 x 16 x 3

#PE	T	Speedup	Efficiency
1*	6105K		
1	6837K	1.00	1.00
2	3431K	1.99	1.00
4	1749K	3.91	.98
8	904K	17.56	.95
16	493K	113.88	.87
32	289K	123.64	.74

* Serial version of the code

Table 1

THREE-DIMENSIONAL WEATHER CODE

24 x 16 x 3

	NUMBER OF INITIALIZATION INSTRUCTIONS				
	10	50	100	500	1000
EFFICIENCY	.98	.91	.83	.50	.33

Table 2

CHAPTER 6

CONCLUSIONS

It is clear from these results that three-dimensional weather calculations are well suited for the Paracomputer and presumably the NYU Ultracomputer.

The amount of surgery on the code that was needed proved to be higher than was initially expected, but was for the most part associated with porting to the CDC-6600, rather that with parallelization per se.

The simulations point up the fact that to attain high efficiency, the number of instructions needed to bring in "helper" processing elements should be kept small since the number of synchronization points per iteration is fairly high. Thus, in a real system, one mode of process creation should take no more that about 100 instructions, not thousands.

Primitive Equations of Motion

1.&2. Horizontal momentum equations

$$V\frac{d\pi}{dt}+\pi\frac{dV}{dt} = \frac{\partial \pi V}{\partial t} + \nabla\cdot(\pi VV) + \frac{\partial}{\partial \sigma}(\pi\dot{\sigma}V)$$

$$= -\pi\nabla\phi - \pi\sigma\frac{RT}{p}\nabla\pi - (f+u\frac{\tan\phi}{a})\,k\times\pi V + \pi F$$

3. Continuity equation

(3.1) $$\frac{\partial \pi}{\partial t} + \nabla\cdot(\pi V) + \frac{\partial}{\partial\sigma}(\pi\dot{\sigma}) = 0 \qquad \text{or}$$

(3.2) $$\frac{\partial \pi}{\partial t} = -\int_0^1 \nabla\cdot(\pi V)\,d\sigma = -\nabla\cdot\int_0^1 \pi V\,d\sigma$$

4. Equation of state

$$\alpha = \frac{RT}{p}$$

5. First law of thermodynamics

$$\frac{\partial \pi T}{\partial t} + \nabla\cdot(\pi VT) + \frac{\partial}{\partial\sigma}(\pi\dot{\sigma}T) = \frac{\pi\omega\alpha}{C_p} + \frac{\pi Q}{C_p} \qquad (\omega=\frac{dp}{dt})$$

From $\theta = T/p^k$, $p=p_T+\sigma\pi$, $\omega=\dot{\sigma}\pi+\dot{\pi}\sigma$, $\dot{\pi}=\frac{\partial\pi}{\partial t}+V\cdot\nabla\pi$, $k=R/C_p$ we get

$$\frac{\partial \pi\dot{\sigma}T}{\partial\sigma} = p^k\frac{\partial \pi\dot{\sigma}\theta}{\partial\sigma} + \pi\frac{\dot{\sigma}\alpha\pi}{C_p} \qquad \text{Replacing in 5,}$$

$$\frac{\partial \pi T}{\partial t} + \nabla(\pi VT) + p^k\frac{\partial \pi\dot{\sigma}\theta}{\partial\sigma} = \frac{\pi\sigma kT}{p}(\frac{\partial\pi}{\partial t} + V\cdot\nabla\pi) + \frac{\pi Q}{C_p}$$

6. Humidity equation

$$\frac{\partial \pi q}{\partial t} + \nabla\cdot(\pi Vq) = 0$$

7. Hydrostatic equation

$$\frac{\partial \phi}{\partial p^k} = -C_p\theta$$

Of the variables $\pi, u, v, T, q, \phi, \alpha, \dot{\sigma}$ we update the 5 primary variables π, u, v, T, and q using equations 1,2,3.2,5 and 6. From equations 3.1, 4, and 7 we can obtain ϕ, α, and $\dot{\sigma}$ which are our secondary variables. Note that $p=\sigma\pi+p_{top}$.

Sea level pressure

$$\text{Hydrostatic eq.} \Rightarrow \frac{\partial p}{\partial \phi} = -p = -\frac{1}{\alpha} = -\frac{p}{RT} \therefore \log\left(\frac{SLP}{p}\right) = -\int_{\phi_s}^{0} \frac{d\phi}{RT}$$

$$\therefore \quad SLP = p(\sigma=1)\ \exp\left(\frac{\phi_s}{R\overline{T}}\right)$$

REFERENCES

[1] Allan Gottlieb, "WASHCLOTH - The Logical Successor to Soapsuds", Ultracomputer Note #12, Courant Institute, N.Y.U., 1980.

[2] Allan Gottlieb, "WASHCLOTH 81", Ultracomputer Note #21, Courant Institute, N.Y.U., 1981.

[3] Allan Gottlieb, "MOP - A (Minimal) Multiprocessing Parallel Operating System", Ultracomputer Note #13, Courant Institute, N.Y.U., 1981.

[4] Allan Gottlieb, B. D. Lubachevsky and Larry Rudolph, "Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors," Ultracomputer Note 16, Courant Institute, N.Y.U., 1980.

[5] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir, "The NYU Ultracomputer --- Designing an MIMD Shared Memory Parallel Computer," IEEE Trans. on Comp., 1983 - to appear.

[6] Eugenia Kalnay-Rivas, "Documentation of the Fourth Order Band Model," NASA Technical Memorandum 80608, Goddard Space Flight Center, 1979.

[7] David Korn, "Converting Scientific Codes to Run Under Washcloth Simulation," Ultracomputer Note 23, Courant

Institute, N.Y.U., 1981.

[8] David Korn, "Timing Analysis for Scientific Codes Run Under Washcloth Simulation," Ultracomputer Note 24, Courant Institute, N.Y.U., 1981.

[9] David Korn, "The DOWAIT and DOALL Implementation," Ultracomputer System Software Note 15, Courant Institute, N.Y.U., 1981.

[10] Borris Lubachevsky, "A Parallel Computer Implementation of Ascend/Descend Type of Vector Algorithms," Ultracomputer Note 37, Courant Institute, N.Y.U., 1981.

[11] Norman Rushfield, "Atmospheric Computations on Highly Parallel MIMD Computers," Ultracomputer Note 22, Courant Institute, N.Y.U., 1981.

[12] J. Schwartz, "Ultracomputers," ACM Transactions on Programming Languages and Systems, Vol. 2, pp. 484-521, 1980.